Survey of Stochastic Number Generators and Optimizing Techniques

Pooja Nahar, Prasad Khandekar, Minal Deshmukh, Harpreet Singh Jatana, and Uday Khambete

Abstract Stochastic computing (SC) is an error-tolerant computing technique where computation takes place on randomized statistical bit streams. The main feature is that standard digital logic gates with low power and low area can be used to implement complex arithmetic functions. However, these advantages need to be compared against slow speed and less precision. Also, the area required by stochastic number generators (SNGs) is high as compared with the area required to implement computation circuit. So, if multiple SNGs are used, then the advantage of low area cannot be extracted. So, there is a need to design good quality, reliable and higher throughput SNGs. Out of different approaches that are used to design SNG, use of low discrepancy sequences and randomization functions has better results with respect to throughput and correlation management, respectively, at the cost of area. So, more research using these two approaches can lead to design of efficient SNG. This paper reviews the background, fundamental concepts, basic arithmetic operations using SC, various approaches to design efficient SNG's by managing correlation in stochastic computing circuits, advantages, and drawbacks followed by conclusion and future challenges.

Keywords Stochastic computing · Approximate computing · Stochastic arithmetic · Random number sources · Random number generator · LFSR · Correlation · Low-power design

P. Nahar (✉)
School of Electronics and Communication Engineering, Dr. Vishwanath Karad MIT World Peace University, Pune, India
e-mail: poojanahargandhi@gmail.com

P. Khandekar · H. S. Jatana
Dr. Vishwanath Karad MIT World Peace University, Pune, India
e-mail: Prasad.khandekar@mitwpu.edu.in

M. Deshmukh
Vishwakarma Institute of Information Technology, Pune, India
e-mail: Minal.deshmukh@viit.ac.in

H. S. Jatana · U. Khambete
Semiconductor Laboratory, Chandigarh, India
e-mail: udayk@scl.gov.in

A. J. Kulkarni et al. (eds.), *Intelligent Systems and Applications*, Lecture Notes in Electrical Engineering 959, https://doi.org/10.1007/978-981-19-6581-4_31

Background

The historical advances in stochastic computing (SC) can be emphasized mainly in four different phases [1]. 1950s were the early years which describe mainly the work done by John V. Neumann. In the 1960s and 1970s, a lot of advancements were made in the technique by researchers like B. R. Gaines, W. J. (Ted) Poppelbaum, Chusin Afuso, John Esch, Sergio Ribeiro, and others who developed the software and the hardware in which SC was used to do the arithmetic operations. The first conference on SC took place in 1978. After that, there was not much research done until 2000. In the year 2001, Howard Card put out two papers on SC in the IEEE [2, 3]. After that, the interest in SC increased. Subsequently, workshops on SC were held at the University of Waterloo in 2016 and in Montreal in 2017. Section 1 of this paper describes the initial work done in SC mainly from the years 1950s to 1970s. Section 2 describes recent development in SC. Section 3 shows errors, ways to minimize these errors, impact of correlation on stochastic computation and ways to manage/manipulate correlation in stochastic circuits. Section 4 covers the optimization techniques for SC circuits. Section 5 and 6 covers the advantages and drawbacks, conclusion and future scope, respectively.

1 Initial Work

There is a lag between application of input and the response at the output of every network. Error is an essential part of such circuit. As shown in [4], this error can be used for synthesizing reliable logic elements using unreliable components for the process under consideration. Similarly, redundancy is also present in biological neural models. This redundancy can be used to compensate component error. Hence, the use of bundles instead of single wire for carrying information was suggested. But, the probability of energizing a wire in a bundle needs to be random. This is because without randomness, error can be amplified instead of getting cancelled out. This bundling of wires can then be used to carry out arithmetic operations like addition (using stochastic multiplexer) and multiplication (based on Sheffer's stroke, like stochastic multiplier). The need for random selection of inputs gave rise to third form of computing apart from digital and analog, which was done using random pulse sequence (RPS) combined with standard logic gates to realize various arithmetic and logical functions. This facilitated huge reduction in area and cost of a computational element.

Afuso as shown in [3, 4] developed RPS for SC. But, since the system required a local random pulse generator, a better system called as synchronous random pulse generator (SRPS) was developed by Afuso under the guidance of Prof. Poppelbaum at the University of Illinois. RPS was discrete; SRPS was digital. Following Fig. 1 shows production of RPS and SRPS. A noise diode is used to generate random noise. The occurrence of noise spikes is then detected by an adjustable detection voltage level and amplified. The output of difference amplifier is sampled by clock. A pulse

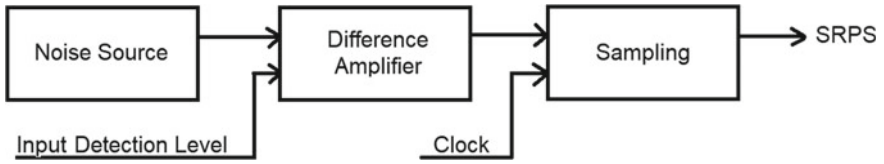

Fig. 1 Simplified SRPS [4]

of standard height and length is generated at the output of the sampling circuit only if it has detected a pulse in the previous slot. This pulse train generated at the output of the sampling device is completely random, and its frequency can be controlled by the detection level. But, buffering in SRPS lead to un-randomizing. So, clocked random pulse sequence (CRPS) was developed. CRPS utilized mapping of numbers in the range $[-1, 1]$ onto range $[0, 1]$ which permitted all operations to be performed with only combinational circuitry, thus preserving the probability distribution.

These RPS was used to design different types of pulse processing systems as seen in [4] like time stochastic processing, bundle processing, ergodic processing, and burst processing. So, all the software algorithms were established and simulated on traditional computer, but applications like machine learning required specific complex hardware structure which was not available at that time. The main challenge being the design of an element which would store and vary the stored value for long periods. The element should also be able to be used as a weight to multiply with other variables. This system needed to occupy less area and low cost because these elements were needed in large numbers in a practical system. To design such a system, analog, electro-chemical devices, or standard digital components using semiconductor devices were the options available. The traditional analog integrates and multiply circuits are not stable and require more cost in terms of area. Electro-chemical devices or transfluxors are not reliable and require refined external circuitry to function properly. The high speed, good stability, small size, low cost, and large-scale integration features of semiconductor devices made them the obvious choice for designing of the computing elements. von Neumann and Pierce [4] shows examples of few stochastic computers developed during the 1960s such as portable stochastic computer (POSTCOMP), adaptive digital element (ADDIE), regular array of stochastic computing element (RASCEL), TRANSFORMATRIX, and autonomous processing element (APE). All these computers used one of the pulse processing systems and perform calculations using probability logic. However, with the advent of large-scale integration (LSI) in the late 1960s, interest in SC decreased until the year 2000. The ability of stochastic circuits to effectively perform tasks like decoding in communications and implementing complex computing tasks at low hardware cost in artificial neural network has rekindled the interest of researchers in this field. Next section describes how SC is used today in computation and various methods in which a SC circuit can be designed and what parameters need to be taken into consideration while doing so.

2 Recent Developments in Stochastic Computing

For performing stochastic computation, the binary number under consideration is first converted into stochastic domain. All the calculations or functions that need to be performed are done in stochastic domain, and the result is then converted back to binary if the required output is supposed to be in binary domain. Following Fig. 2 [5] shows a traditional stochastic number generator (SNG). It is used to convert a binary number into a stochastic number and vice versa. A comparator is used as a probability conversion circuit (PCC) that compares the binary number of m bits with a random number of m bits generated by the random number source (RNS). For every binary number B greater than R, a one is produced at the output. The number X is called as a stochastic number (SN) and has a probability pX. It is defined as the ratio of total number of 1s to total number of bits in a stochastic bit stream. E.g., A sequence $X = 10{,}101{,}100$ has Px = Probability of 1s in X as (4/8), i.e., 0.5. All the arithmetic operations are done in the stochastic domain using the generated SN's. The final output can then be converted to binary format by using a counter that will count the number of 1s in the sequence X. The randomness and length N determine the accuracy of created SN. Ideally, a stochastic number generator should be able to generate random binary sequence where each new bit is independent of the previous bit. Various RNS have been used like noise diode, LFSR [6], NLFSR [7], nanodevices like magnetic tunnel junction (MTJ) [8], and memristors. Some of which are true RNS, while others are pseudo-RNS. Similar to one in Fig. 2, there have been different RNG's that are designed like weighted binary RNG as proposed by Gupta and Kumaresan in [6], 4-bit binary SNG by Van Daalen et al. [9], and sampling-based RNG [10].

Using above circuit, all the basic arithmetic operations, operations using memory and design of finite state machines can be done as shown in Fig. 3. Multiplication is performed using an AND gate. Addition (scaled) is done using a multiplexer. Similarly, division is performed by using a counter in the specific arrangement using D flip-flop, and subtraction is performed using an EX-OR gate. The range of the inputs should be between [0, 1] to be able to be represented as a probabilistic bit

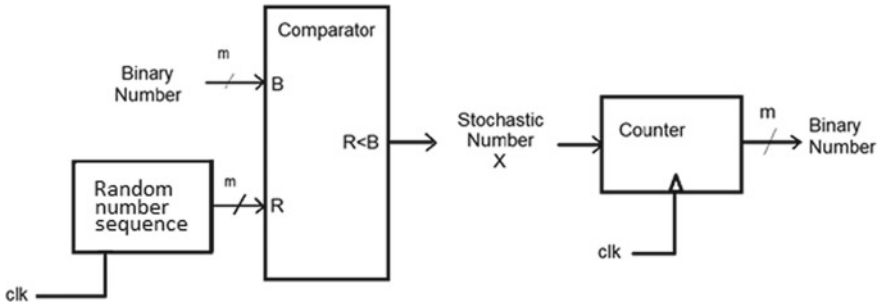

Fig. 2 Binary to stochastic and stochastic to binary converter [5]

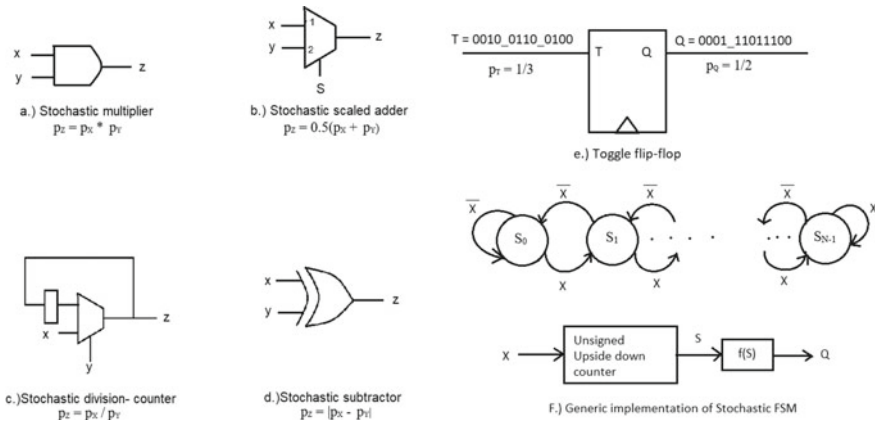

Fig. 3 Arithmetic, sequential, and FSM operations using SC [1] **a** multiplication, **b** scaled addition **c** division **d** subtraction **e** sequential element—flip-flop **f** FSM

sequence. If not, the inputs need to be scaled. To represent a polynomial or non-polynomial function such as trigonometric functions whose co-efficient do not lie in the interval [0, 1], they are converted to Bernstein polynomials. For operations using memory, consider a toggle flip-flop (TFF). The TFF has the output probability of 1/2 irrespective of any non-zero input. So, TFF can be used to generate a stochastic number with a fixed probability thus ruling out the requirement of an RNG. Finite state machine (FSM) implementation of SC is shown in Fig. 3f where an up/down counter is used to obtain the state-transition. The counter will increment at $X = 1$ and decrement for $X = 0$. F is the mapping of output which can be realized using a combinational function of S.

3 Errors and Correlation in Stochastic Computing

Stochastic circuits can be designed using two different approaches. One is reconfigurable stochastic circuits, and other is fixed stochastic circuits. The inputs of reconfigurable stochastic circuits are programmable, and the circuit can be reused to implement different functions, whereas fixed stochastic circuits can be used for a dedicated function implementation. But, since SC is an approximate form of computing, there are various known sources of errors [1] as shown in Fig. 4.

In above circuit, C is the core of the stochastic circuit. It can either be sequential or combinational logic circuit. $X1$, $X2$…. Xn are the input stochastic numbers of length N and are assumed to be independent of each other. To convert the input X to a desired probability of the value of X, the ancillary inputs $R1$, $R2$,…..Rk with constant value $Ri = 0.5$ and are used. Rk's and Xn's are assumed to be independent of each other. Any physical errors caused due to defect in hardware or sources like

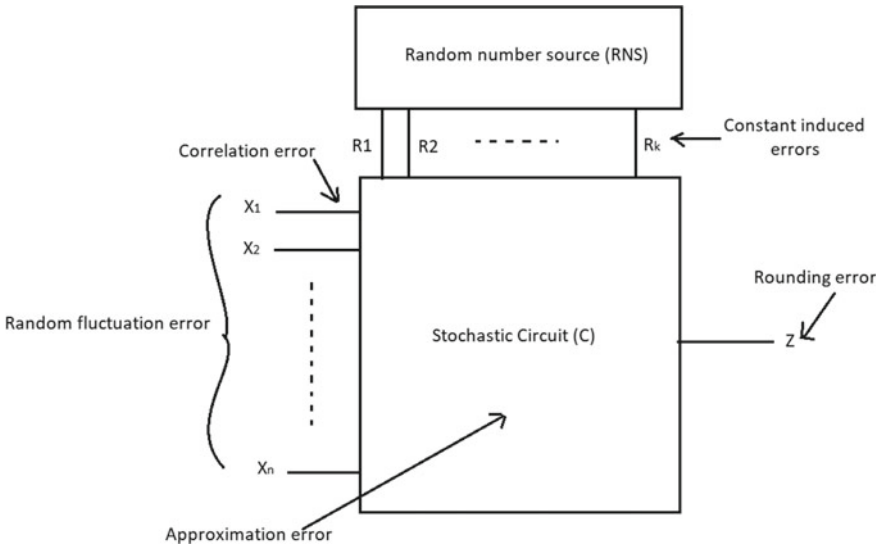

Fig. 4 Generic stochastic circuit with known sources of errors [1]

radiation are not shown in Fig. 4. The errors and the ways to overcome errors are described in detail as below:

Rounding Errors: These are errors caused due to quantization. For, e.g., a bit stream of Y bits can constitute of $(Y + 1)$ numbers as SN $= \{0, 1/Y, 2/Y, ..., (Y - 1)/Y, 1\}$. If a number under consideration Z is not in this set, then it is rounded to the closest value in the set of SN. For example, with $Y = 8$ and $Z = 0.130$, we must round off Z to $1/8 = 0.125$. The precision can be increased by increasing the length of SN.

Approximation Errors: These errors occur because not all functions can be realized exactly by SC. It is necessary to approximate them to something that is achievable. All stochastic function values are scaled to be in the interval [0, 1]. Without Rk's as inputs, stochastic functions using one variable that are synthesizable are only trivial case of X and $1-X$. So, functions like square root, squaring, $\sin(X)$ can be done by using Bernstein approximation.

Random Fluctuation Errors: These errors are caused because the N bit SN formed by using an RNG is pseudo-random in nature. These errors can be specified by calculating the mean square error. As the length of stochastic bit stream is increased, random fluctuation errors and rounding errors tend to decrease.

Constant-Induced Errors: These errors are caused because of the ancillary inputs. However, these errors can be reduced by transferring the ancillary inputs to sequential sub-circuits C so that their behavior can be traced. An algorithm CEASE is established to take out this error. CEASE replaces the ancillary inputs by sequential

components equal to a mod counter with multiple moduli. It releases the weighted input bits when the counter-overflows.

Correlation Errors: Errors caused by correlation are due to intercommunication between the bit streams that occurs during computation which introduce dependencies and similarities between different bit streams. Correlation is also caused mainly due to three factors: RNS which are not able to produce good quality independent SNs, sharing of RNS to decrease the hardware cost of circuit and temporal dependencies introduced by sequential circuits. So, as the circuit size increases, correlation also increases. To quantify correlation, the stochastic computing (SCC) for a pair of SNs, X and Y, is defined as follows in Eq. 1:

$$\mathrm{SCC}(X,\ Y) = \begin{cases} \frac{p_{X\wedge Y} - p_X \cdot p_Y}{\min(pX,\ pY) - pXpY} & \text{if } p_{X\wedge Y} > p_X \cdot p_Y \\ \frac{p_{X\wedge Y} - p_X \cdot p_Y}{pXpY - \max(pX + pY) - 1,\ 0} & \text{otherwise} \end{cases} \tag{1}$$

where $P_{X\wedge Y}$ is $P(X(t) = 1, Y(t) = 1)$ for all t. Bit streams that are positively correlated, i.e., the 1s and 0s of a bit stream overlap with each other have SCC value as 1. Negatively correlated means that the 1s and 0s of the bitstreams under consideration do not overlap at all, and SCC value is -1. Non-correlated or independent bit streams are the ones where each bit stream is obtained from different Bernoulli sequences as the random number source giving SCC value as 0. There are several ways to avoid or manipulate correlation as below:

(a) **Regeneration**: In this method, the corrupted SN is converted into binary format and then given to a SNG to generate a new SN from it. This method has added to more hardware, more latency and desirable properties like progressive precision can be lost. [11]. Figure 5a shows a regeneration-based decorrelator.
(b) **Isolation**: In this method, D flip-flop is introduced at the bit stream frequency into the line containing corrupted SN. So, it delays the corrupted SN by one clock cycle. If the corrupted SN under consideration has bits which are independent, naturally, the new delayed version of the corrupted SN and the original SN are independent of each other. Figure 5b shows isolation-based decorrelator.
(c) **Shuffle-based Decorrelation**: Shuffling will re-randomize the bits in the corrupted SN thus helping reduce cross-correlation and autocorrelation. Complete elimination of correlation is not possible by this method because the newly generated SN will be correlated with original SN. Figure 5c shows an example of shuffle-based decorrelation.

(a) **Correlation Insensitive**: There are few examples of stochastic circuits which are not sensitive to correlation. It means, there is no need to bother about the auto- or cross-correlation in such circuits. It means, the circuit will function as desired even with correlated inputs. Such stochastic computing circuits are called as correlation-insensitive circuits.
(b) **Correlation Injection by Synchronizer and Desynchronizer**: Many stochastic circuits require uncorrelated inputs. But, circuits that implement functions which require the SN's to be either positively or negatively correlated. E.g.,

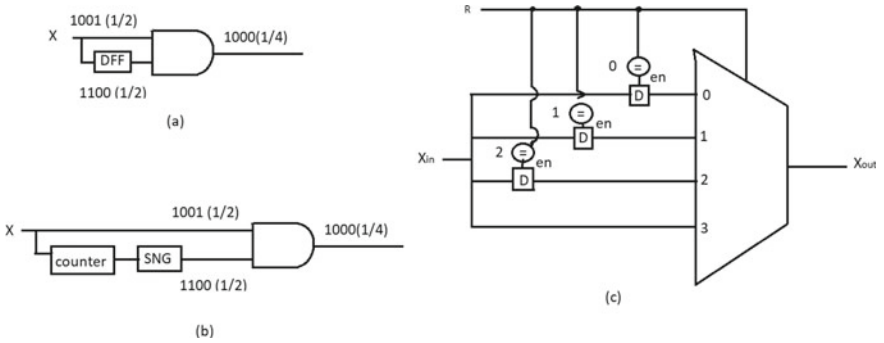

Fig. 5 **a** Regeneration-based decorrelator [11], **b** Isolation-based decorrelator [11], **c** Shuffle-based decorrelator [1]

to calculate the difference, an Ex-OR gate requires maximally correlated inputs. A type of SNG called synchronizer that can control the amount of correlation is used in such circuits. But, regenerating SNs with desired correlation levels between a stochastic system incurs a high hardware cost and introduces latency. Similarly, a desynchronizer can be used to decrease the correlation between the SN's. Errors due to autocorrelation can be managed by CEASE.

4 Optimization of Stochastic Circuits

Many attempts are being made to optimize the SC circuits with respect to various factors. The random number generator is one of the main blocks of SC which occupies more hardware area and needs to be optimized. The second part is correlation, which if managed properly can save a lot of hardware area, decrease latency, and increase the accuracy. The third part is the connecting wires between RNS and PCC. Various experimentations with respect to RNG have been done in various research papers until now. They are listed as below:

(i) Circular shifting of output of LFSR to be able to use a shared RNG [12].
(ii) Permutation of output of shared LFSR in reverse lexicographic order instead of circular shifting [13].
(iii) Use of S-box at the LFSR's output and then share it to multiple RNGs. This method increases hardware complexity as the S-box is a complex combinational circuit [14].
(iv) LFSR with nonlinear S-box function to show better autocorrelation and cross-correlation properties as compared to LFSR [15].
(v) Use of spintronic devices instead of LFSR to decrease the area and power consumed by the stochastic core [16].
(vi) Designing SNG's by sharing an RNS based on randomization function [17].

(vii) Use of LUT-based Quasi SNG where low discrepancy (LD) sequences are used to develop stochastic numbers [18]. Since LD sequences are uniformly spaced 1 s and 0 s, this SNG has better accuracy, convergence, throughput as compared to LFSR, and a less random fluctuations and execution time because of parallel implementation. The drawbacks include need for optimization with respect to area.

(viii) Bit flipping of output of LFSR that makes the generated SN's correlation insensitive and allows sharing of RNS [19]

Similarly, correlation can be managed by using correlation-insensitive circuits or correlation manipulative circuits like isolators, synchronizers, desynchronizer, decorrelators, regenerators, or judiciously selecting the seed of LFSR for generating RNS.

Considering above all design aspects, SC has been applied to applications where many arithmetic operations need to be done. This is because, simple logic gates are used to perform these operation, and hence, resultant decrease in the area and power is in 100 s as compared to traditional deterministic computing. Also, since the precision required in such applications is low, use of excessively long SN's can be avoided thus limiting the latency.

5 Advantages and Drawbacks

SC utilizes randomness as a valuable resource and converts probability values into another statistical bit stream. It lacks place value, i.e., no LSB or MSB as compared to deterministic computing. This makes the computation robust. Apart from this, other advantages of SC are small area, less power, error tolerant, supports parallelism; standard digital components can be used to realize the circuits. Its primary disadvantages are correlation induced inaccuracy, random number fluctuations, high latency; randomness sources are costly, and the theory is not yet fully understood. The design of a stochastic circuit involves complex trade-offs among accuracy, computing time, and hardware area cost.

6 Conclusion and Future Challenges

From the various methods to design a SNG described above, use of LD sequences for generating RNS gives higher throughput. Also, use of randomization function at the output of LFSR gives better correlation management. However, both of these methods claim additional area cost. So, use of either or both of above methods to design SNG without the addition of area or latency overhead need to be explored more for future work.

Also, automatic synthesis of optimally correlated deterministic RNS for design SNG's is also a challenge.

The domains to which SC can be successfully applied are artificial neural networks (ANNs), image processing, control systems, DSP, and decoding of modern error correcting codes [5]. Application of SC in implantable medical devices still remains a challenge for the researchers.

References

1. Gaudet VC, Gross WJ, Smith KC (2019) Stochastic computing: techniques and applications. Springer, Cham
2. Afuso C (1968) Analog computation by random pulse sequences. University of Illinois, Urbana, Illinois, Department of Electrical engineering
3. Poppelbaum WJ (1976) Statistical processors. In: Rubinoff M, Yovits MC (ed) Advances in computers, vol 14. Elsevier, Department of Computer Science University of Illinois Urbana, Illinois, pp187–230. Available: https://doi.org/10.1016/S0065-2458(08)60452-0
4. von Neumann J, Pierce RS (1952) Lectures on probabilistic logics and the synthesis of reliable organisms from unreliable components. Delivered at the California Institute of Technology, January 4–15, Caltech Eng. Library, QA. 267.V6
5. Alaghi, Hayes JP (2013) Survey of stochastic computing. ACM Trans Embed Comput Syst 12(2):1–19
6. Gupta PK, Kumaresan (1988) Binary multiplication with PN sequences. IEEE Trans Acoust Speech Sig Process 36:603–606
7. Sanath Potnuru SRPVS, Venkatesh TS, Agrawal S (2021) Design and implementation of efficient stochastic number generator. In: Fourth international conference on electrical, computer and communication technologies (ICECCT), pp 1–7
8. Perach B, Kvatinsky S (2020) An asynchronous and low-power true random number generator using STT-MTJ. In: 2020 IEEE international symposium on circuits and systems (ISCAS)
9. Van Daalen M, Jeavons P, Shawe-Taylor J, Cohen D (1993) Device for generating binary sequences for stochastic computing. Electron Lett 29:80
10. Karadeniz MB, Altun M (2017) Sampling based random number generator for stochastic computing. In: 2017 24th IEEE international conference on electronics, circuits and systems (ICECS), pp 227–230
11. Alaghi, Qian W, Hayes JP (2018) The promise and challenge of stochastic computing. IEEE Trans Comput-Aided Des Integr Circuits Syst 37(8):1515–1531
12. Ichihara H, Ishii S, Sunamori D, Iwagaki T, Inoue T (2014) Compact and accurate stochastic circuits with shared random number sources. In: 2014 IEEE 32nd international conference on computer design (ICCD), pp 361–366
13. Salehi SA (2019) Low-correlation low-cost stochastic number generators for stochastic computing. In: 2019 IEEE global conference on signal and information processing (GlobalSIP), pp 1–5
14. Neugebauer F, Polian I, Hayes JP (2018) S-box-based random number generation for stochastic computing. Microprocess Microsyst 61:316–326, Elsevier
15. Neugebauer F, Polian I, Hayes JP (2017) Building a better random number generator for stochastic computing. In: Euromicro conference on digital system design (DSD), pp 1–8
16. Angizi S, He Z, Bai Y et al (2018) Leveraging spintronic devices for efficient approximate logic and stochastic neural networks. In: Proceedings of 2018 great lakes symposium on VLSI (GLSVLSI'18). ACM, New York, p 6
17. Tawada M, Togawa N (2020) Designing stochastic number generators sharing a random number source based on the randomization function. In: 2020 18th IEEE international new circuits and systems conference (NEWCAS), pp 271–274

18. Seva R, Metku P, Choi M (2017) Energy efficient FPGA based parallel Quasi-stochastic computing. J Low Power Electr Appl
19. Abdellatef H, Khalil-Hani M, Shaikh-Husin N et al (2021) Low-area and accurate inner product and digital filters based on stochastic computing. In: Signal processing, vol 183